Heap min-max

Heap min-max

Acessa o máximo e o mínimo de uma lista em $\Theta(1)$.

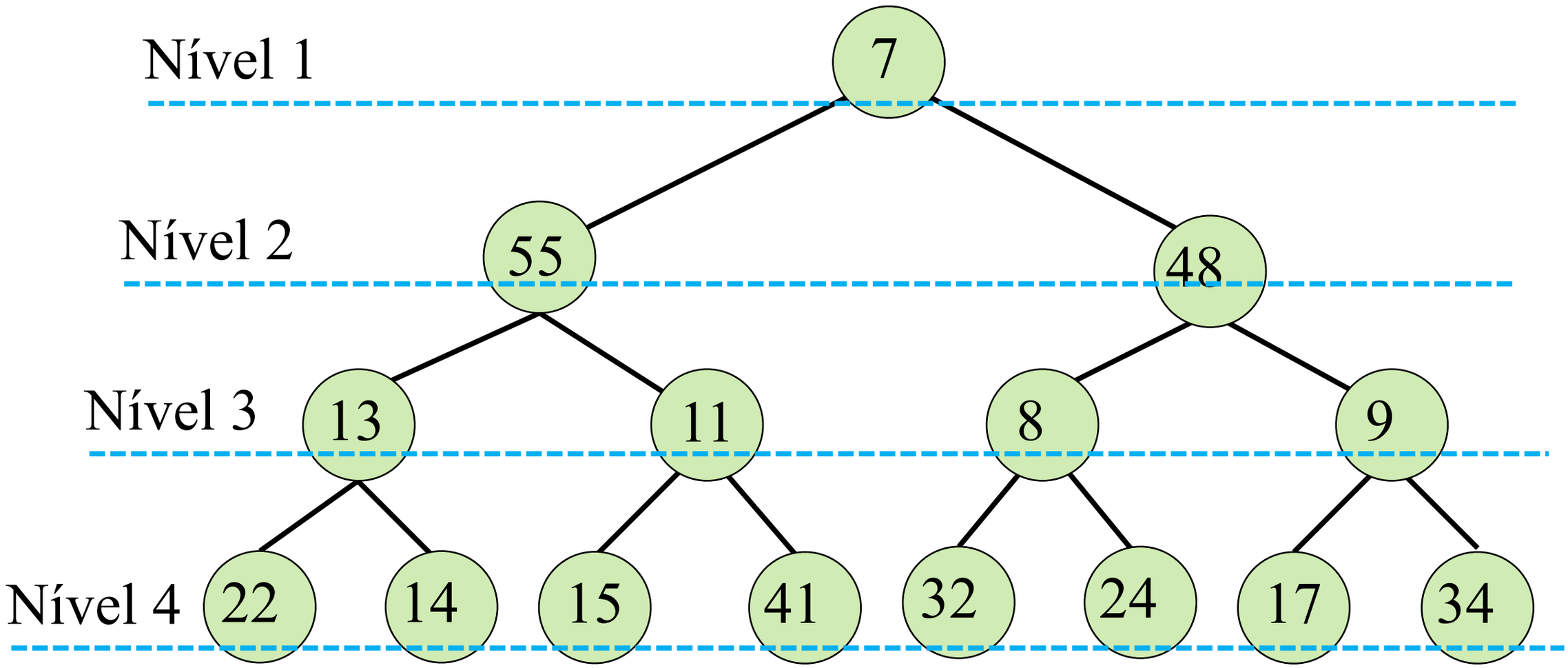
Dado um elemento s_i, $1 \leq i \leq n$, o *nível* de s_i (chave da *i*-ésima posição) é:

$$\text{nível de } s_i = \lfloor \log i \rfloor + 1$$

Nível par é nível de *máximo*

Nível ímpar é nível de *mínimo*

Heap min-max



7	55	48	13	11	8	9	22	14	15	41	32	24	17	34

Heap min-max

- s_1 é a menor chave

Para toda s_i, $1 < i \leq n$:

- i está em nível ímpar (mínimo)

$$s_i \leq s_{\lfloor i/2 \rfloor}$$

$$s_i \geq s_{\lfloor i/4 \rfloor}, \; i \geq 4$$

***i* deve ser menor do que seu pai e maior que seu avô**

- i está em nível par (máximo)

$$s_i \geq s_{\lfloor i/2 \rfloor}$$

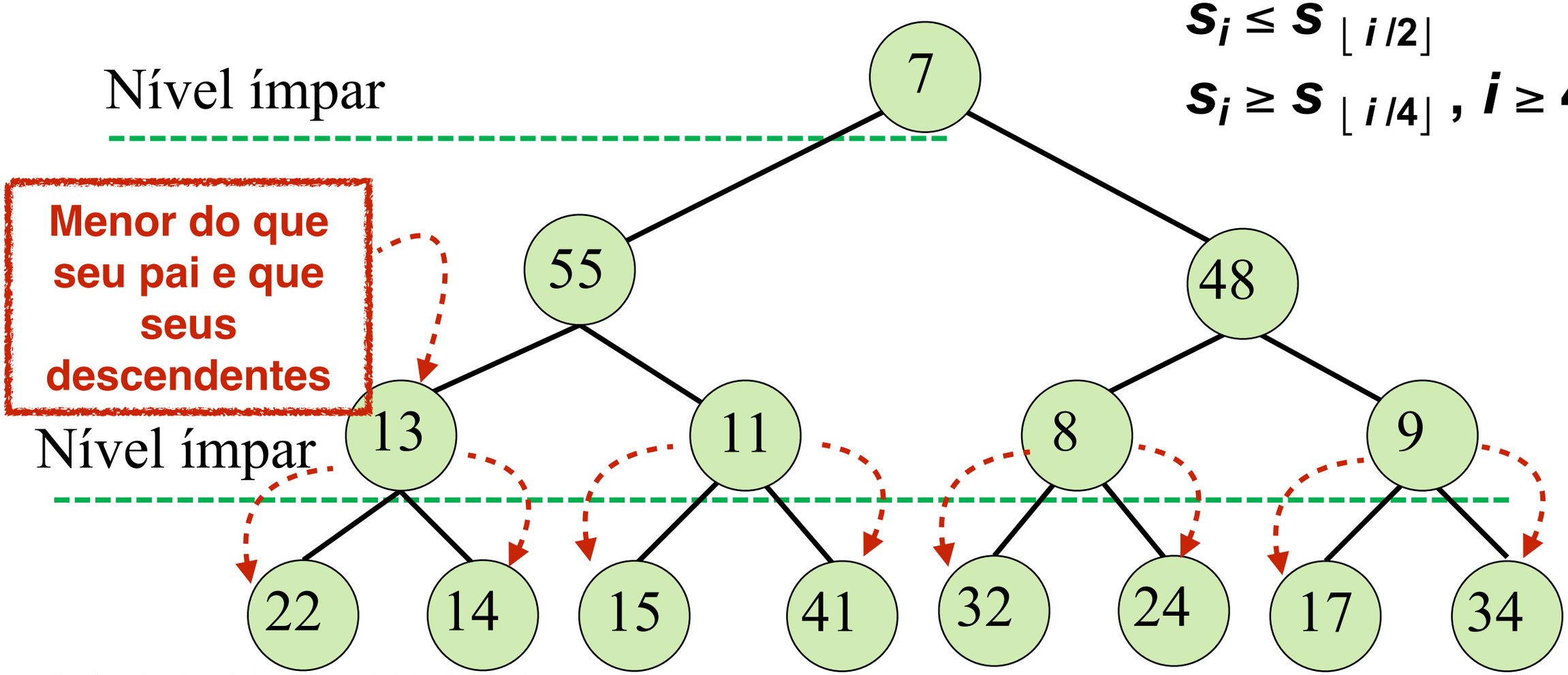
$$s_i \leq s_{\lfloor i/4 \rfloor}, \; i \geq 4$$

***i* deve ser maior do que seu pai e menor que seu avô**

Heap min-max

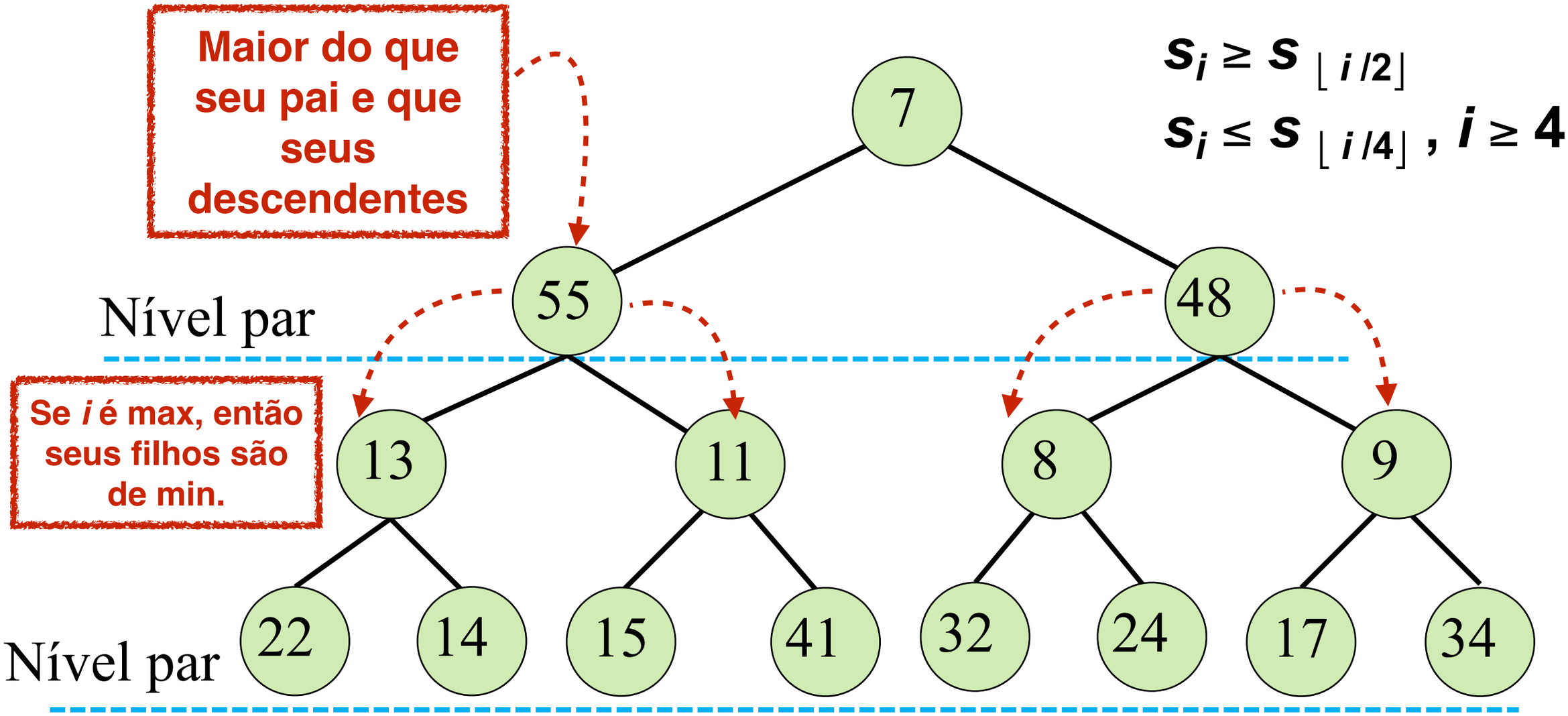
$s_i \leq s_{\lfloor i/2 \rfloor}$

$s_i \geq s_{\lfloor i/4 \rfloor}$, $i \geq 4$



7	55	48	13	11	8	9	22	14	15	41	32	24	17	34

Heap min-max



7	55	48	13	11	8	9	22	14	15	41	32	24	17	34

Heap min-max

Procedimentos

Remover mínimo

Remover máximo

Inserção

Construção

Heap min-max

Remover máximo

Seja *i* o índice do maior filho da raiz

Substituir o *i*-ésimo pelo último elemento da lista e executar *descer*(*i*).

A verificação do nó a ser trocado com *i* no procedimento *descer*( ) deve ser feita entre os filhos e netos do nó *i*.

Heap min-max

```
descer(i)
   se nível(i) = "min" então
      descer_min(i)
   senão descer_max(i)
```

Heap min-max

```
descer_min(i)
   se H[i] tem filhos então
      m ← índice do menor dos filhos e netos de i
      se H[m].chave < H[i].chave então
            Troca(H[m], H[i])
      se H[m] é neto de H[i] então
         pai ← ⌊m/2⌋
         se H[m].chave > H[pai].chave então
            Troca(H[m], H[pai])
         descer_min(m)
```

O elemento *i* deve ser menor que seus filhos e seus netos

Se *H[m]* é neto de *H[i]*, então *H[m]* é de mínimo e *H[pai]* é de máximo

***H[pai]* deve ser maior do que seus filhos e seus netos.**

Obs: O procedimento descer_max é similar, exceto pela inversão das relações < e >.

Heap min-max

```
descer_max(i)
   se H[i] tem filhos então
      m ← índice do maior dos filhos e netos de i
      se H[m].chave > H[i].chave então
            Troca(H[m], H[i])
      se H[m] é neto de H[i] então
            pai ← ⌊m/2⌋
            se H[m].chave < H[pai].chave então
                Troca(H[m], H[pai])
            descer_max(m)
```

O elemento *i* deve ser maior do que seu pai, que seus filhos e seus netos

Se *H[m]* é neto de *H[i]*, então *H[m]* é de máximo e *H[pai]* é de mínimo

***H[pai]* deve ser menor do que seus filhos e seus netos.**

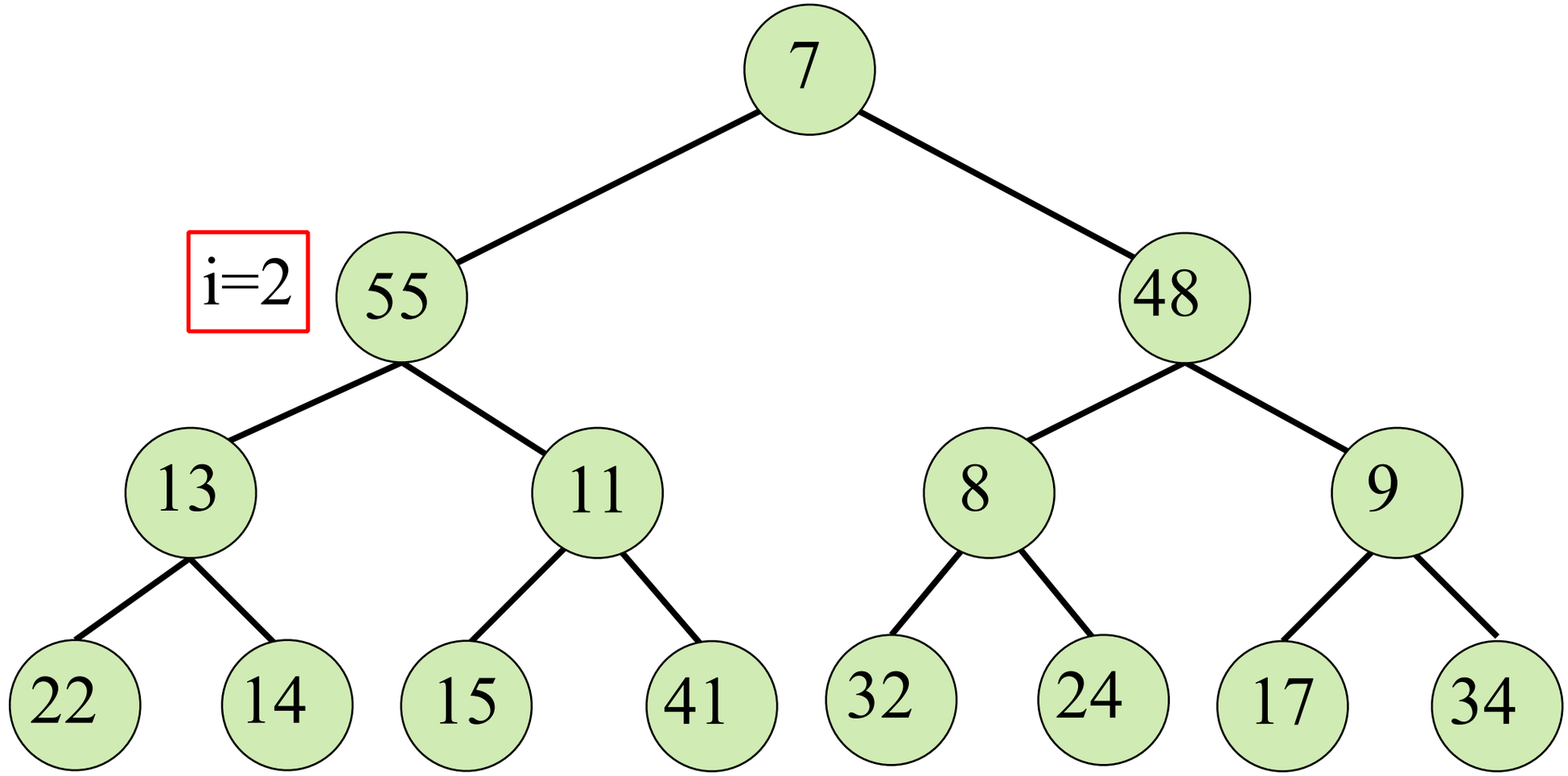
Heap min-max

Exemplo 1: Remover máximo

→ **Seja *i* o índice do maior filho da raiz**

Substituir o *i*-ésimo pelo último elemento da lista e executar *descer*(*i*).

A verificação do nó a ser trocado com *i* no procedimento *descer*( ) deve ser feita entre os filhos e netos do nó *i*.



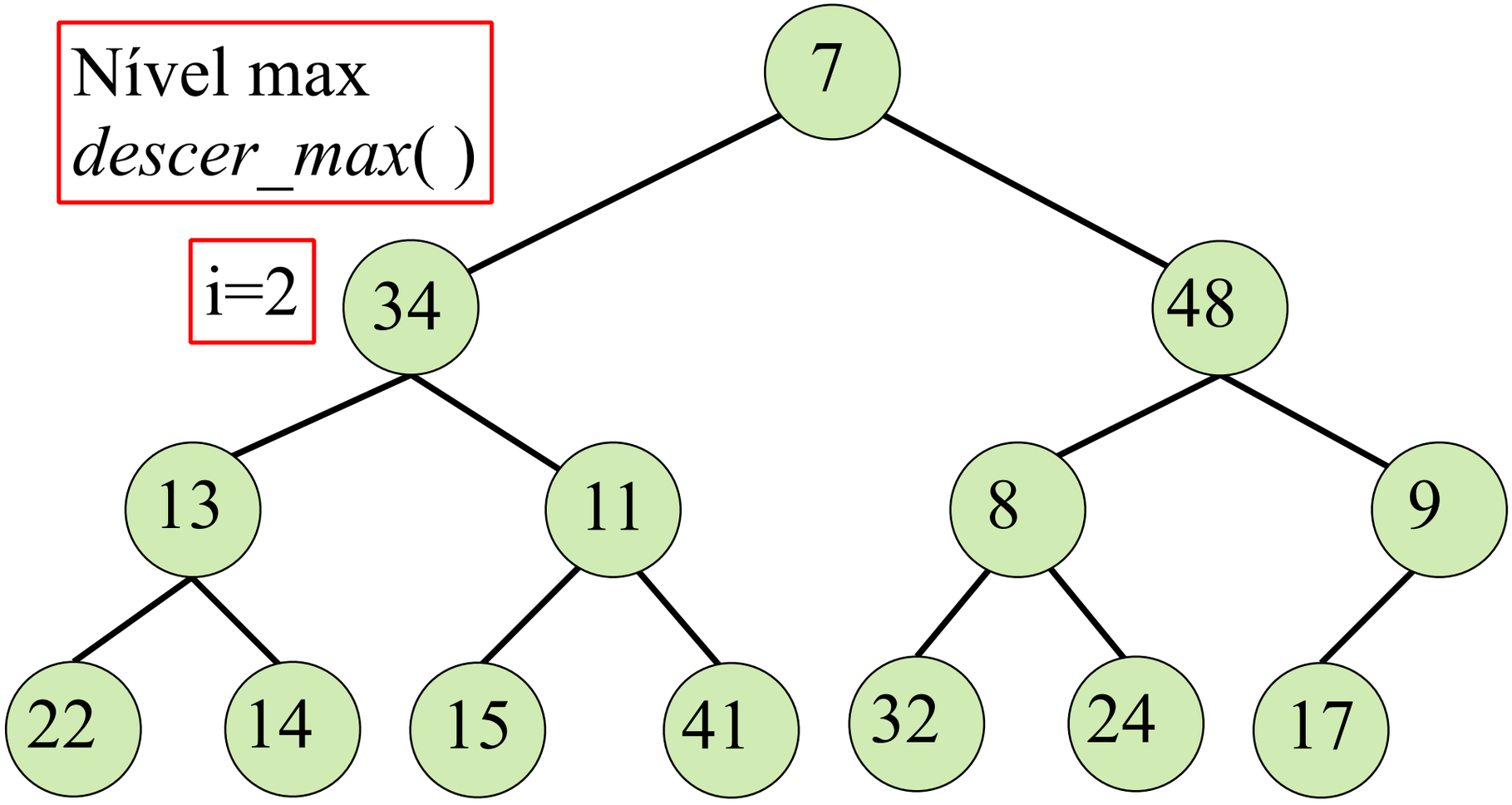
Heap min-max

Exemplo 1: Remover máximo

Seja *i* o índice do maior filho da raiz

Substituir o *i*-ésimo pelo último elemento da lista e executar *descer*(*i*).

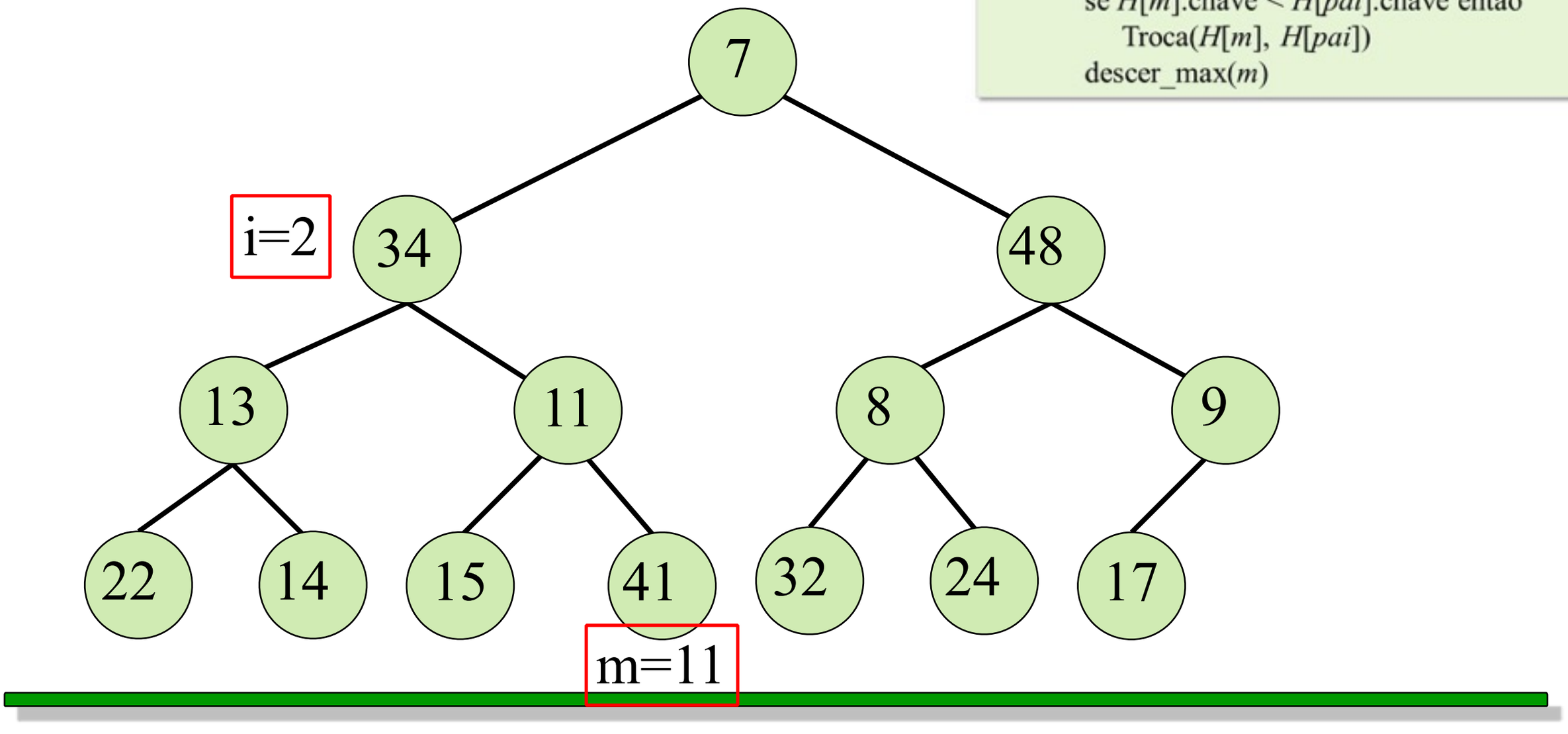
A verificação do nó a ser trocado com *i* no procedimento *descer*( ) deve ser feita entre os filhos e netos do nó *i*.



Heap min-max

Exemplo 1: Remover máximo

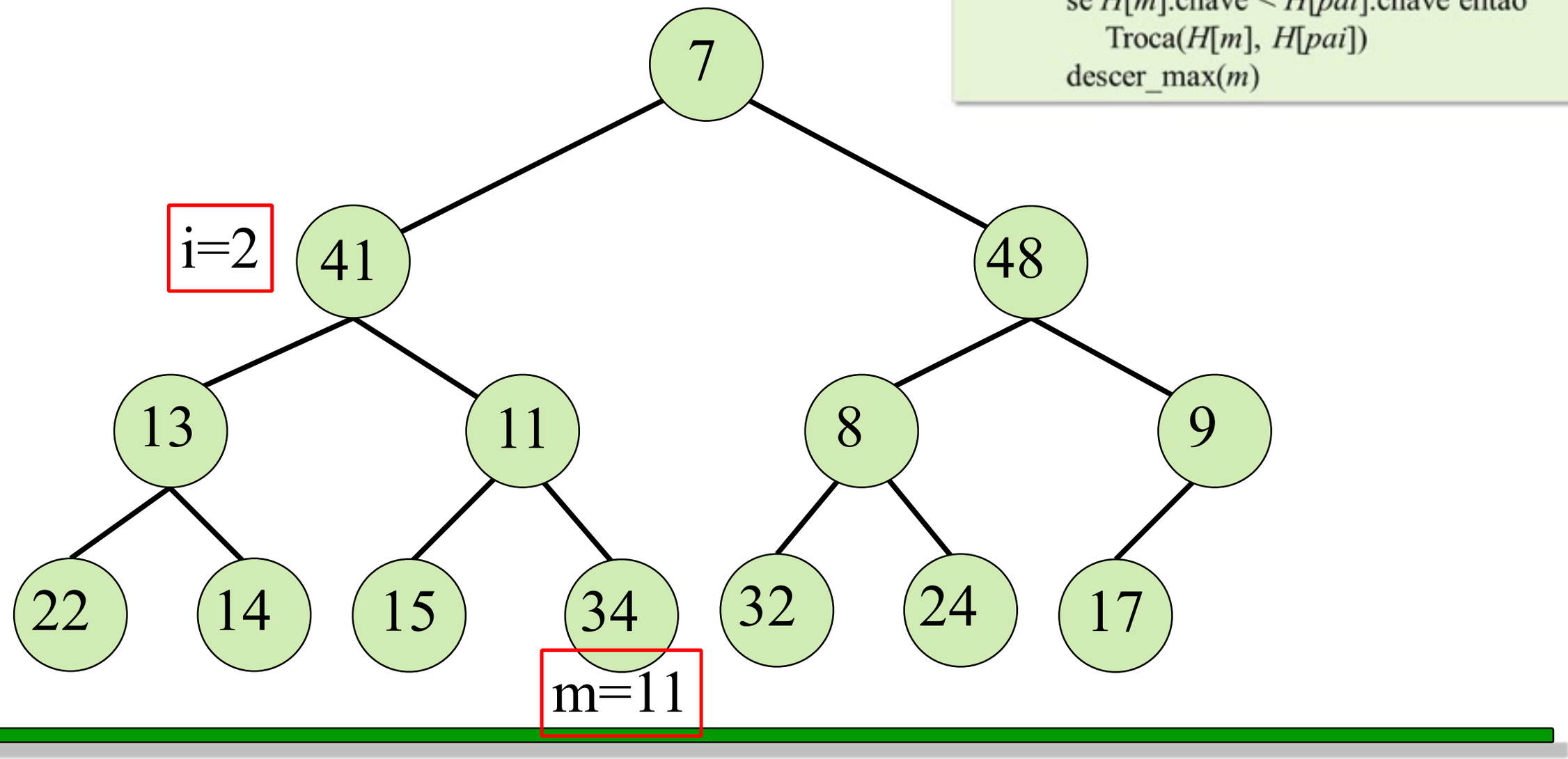
```
descer_max(i)
  se H[i] tem filhos então
     m ← índice do maior dos filhos e netos de i
     se H[m].chave > H[i].chave então
          Troca(H[m], H[i])
     se H[m] é neto de H[i] então
          pai ← ⌊m/2⌋
          se H[m].chave < H[pai].chave então
               Troca(H[m], H[pai])
          descer_max(m)
```



Heap min-max

Exemplo 1: Remover máximo

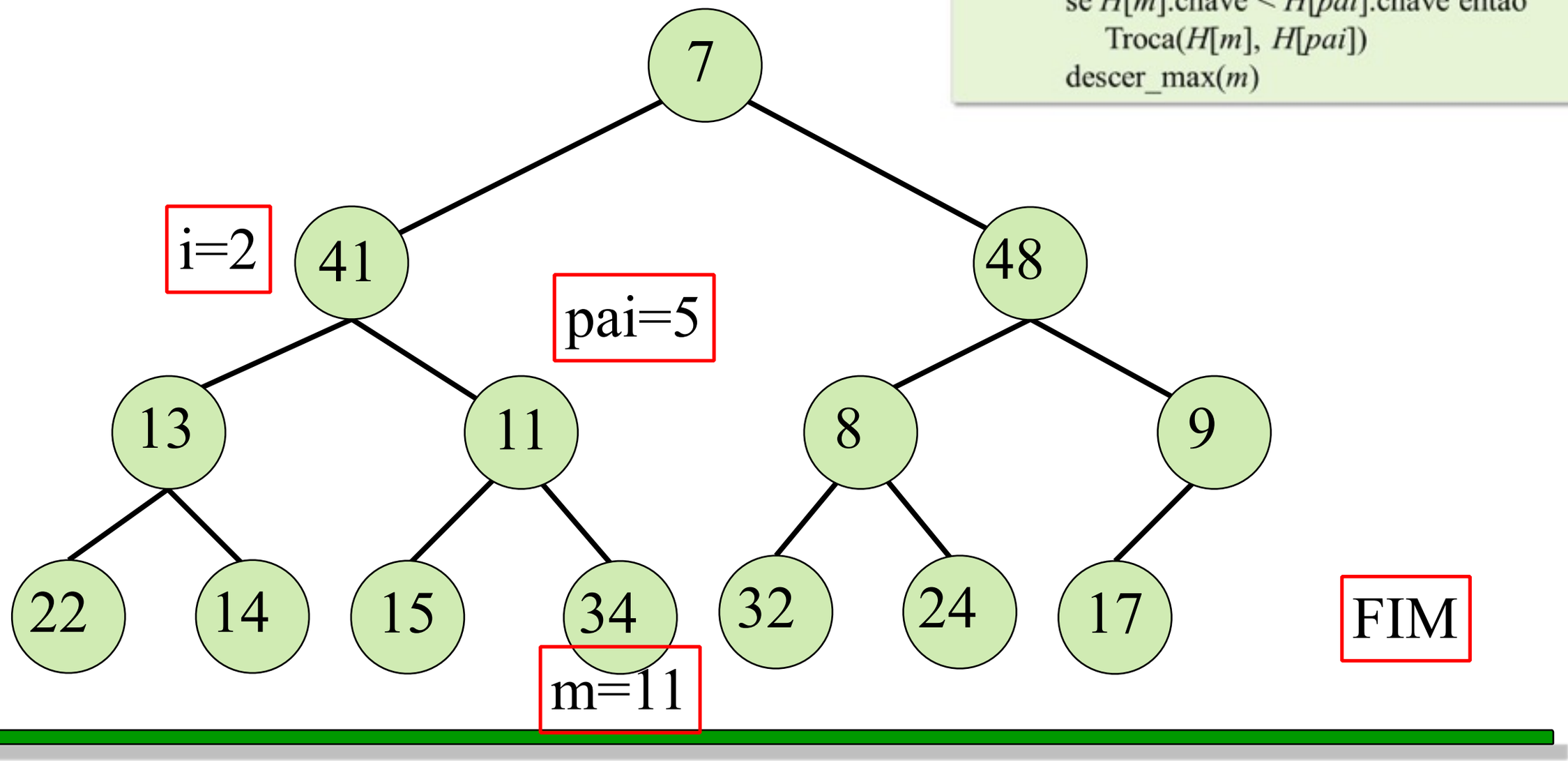
```
descer_max(i)
  se H[i] tem filhos então
    m ← índice do maior dos filhos e netos de i
    se H[m].chave > H[i].chave então
        Troca(H[m], H[i])
    se H[m] é neto de H[i] então
        pai ← ⌊m/2⌋
        se H[m].chave < H[pai].chave então
            Troca(H[m], H[pai])
        descer_max(m)
```



Heap min-max

Exemplo 1: Remover máximo

```
descer_max(i)
  se H[i] tem filhos então
    m ← índice do maior dos filhos e netos de i
    se H[m].chave > H[i].chave então
        Troca(H[m], H[i])
    se H[m] é neto de H[i] então
        pai ← ⌊m/2⌋
        se H[m].chave < H[pai].chave então
            Troca(H[m], H[pai])
        descer_max(m)
```



Exercício

2. Escrever o procedimento para remover o mínimo.

Heap min-max

Inserção

Incluir a chave na última posição, *i*.

Comparar a chave com a chave do pai.
(Se *i* está em um nível de mínimo, o pai está em um máximo e vice-versa).

Havendo ou não troca entre *i* e seu pai, o procedimento de subida deve ser efetuado de acordo com o nível que se encontra o novo valor.

Heap min-max

```
subir(i)
   pai ← ⌊i/2⌋
   se nível(i) = "min" então
      se pai ≥ 1 então
         se H[i].chave > H[pai].chave então
            Troca(H[i], H[pai])
            subir_max(pai)
         senão subir_min(i)
   senão
      se pai ≥ 1 então
         se H[i].chave < H[pai].chave então
            Troca(H[i], H[pai]);
            subir_min(pai)
         senão subir_max(i)
```

Então o elemento *i* deve ser menor que seu pai.

O elemento *pai*, que recebeu o valor do filho *i*, está no nível de máximo

Então o elemento *i* deve ser MAIOR que seu pai.

Heap min-max

i está no nível de mínimo

i deve ser maior que seu avô

```
subir_min(i)
    se H[i] tem avô então
        se H[i].chave < H[avô].chave então
            Troca(H[i], H[avô])
            subir_min(avô)
```

Lembrete:

$s_i \leq s_{\lfloor i/2 \rfloor}$

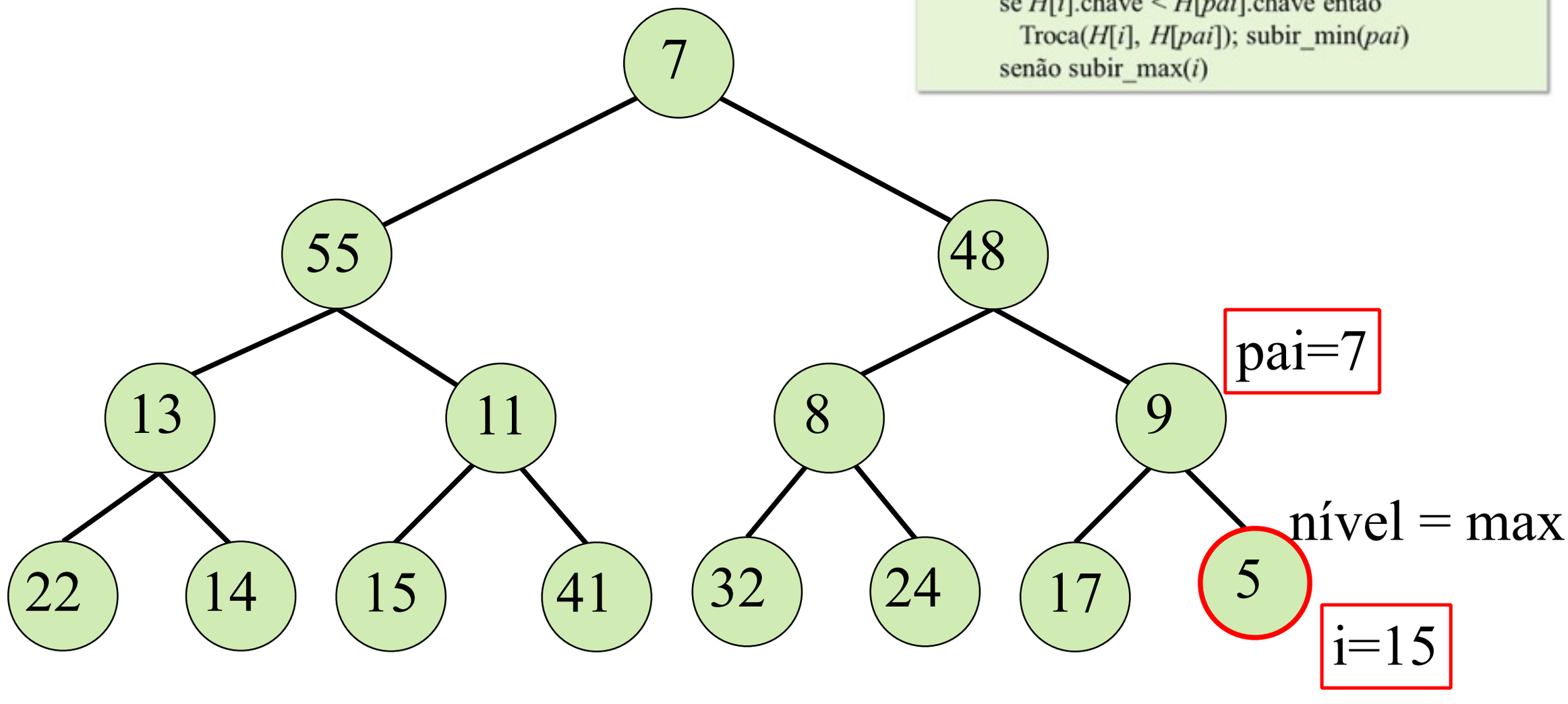
$s_i \geq s_{\lfloor i/4 \rfloor}$, $i \geq 4$

Obs: O procedimento subir_max é similar, exceto pela inversão da relação <.

Heap min-max

Exemplo 2: Inserção

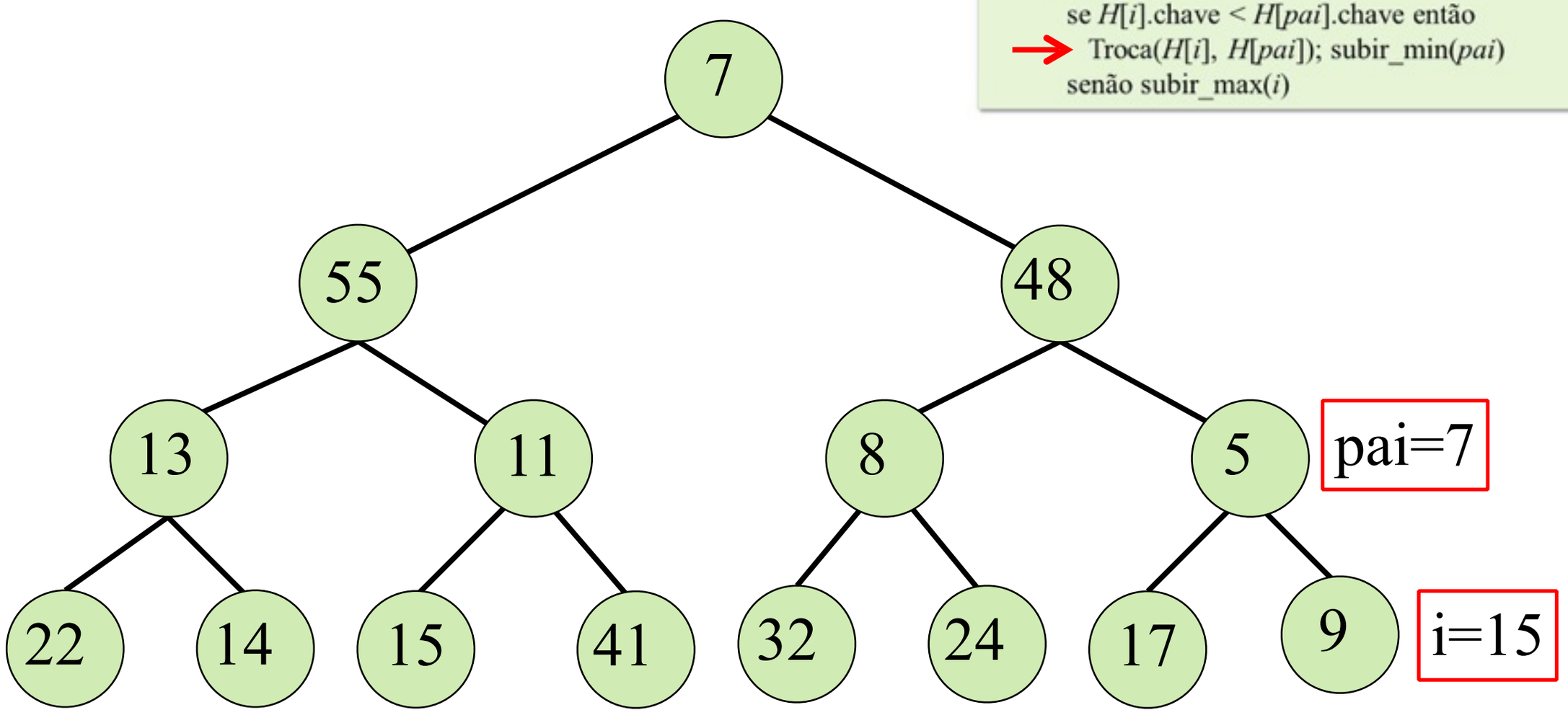
```
subir(i)
  pai ← ⌊i/2⌋
  se nível(i) = "min" então
     se pai ≥ 1 então
        se H[i].chave > H[pai].chave então
           Troca(H[i], H[pai]); subir_max(pai)
        senão subir_min(i)
  senão
     se pai ≥ 1 então
        se H[i].chave < H[pai].chave então
           Troca(H[i], H[pai]); subir_min(pai)
        senão subir_max(i)
```



Heap min-max

Exemplo 2: Inserção

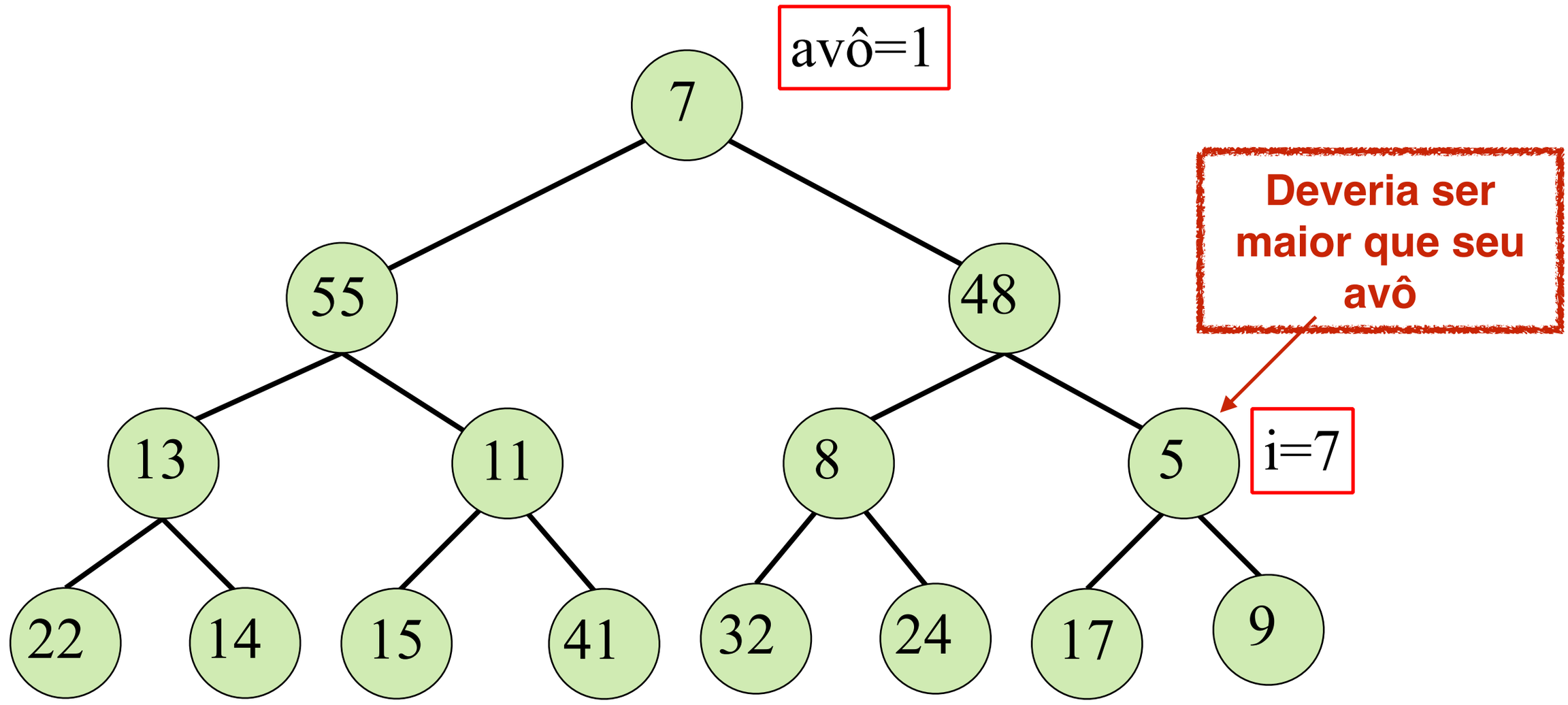
```
subir(i)
  pai ← ⌊i/2⌋
  se nível(i) = "min" então
     se pai ≥ 1 então
        se H[i].chave > H[pai].chave então
           Troca(H[i], H[pai]); subir_max(pai)
        senão subir_min(i)
  senão
     se pai ≥ 1 então
        se H[i].chave < H[pai].chave então
        → Troca(H[i], H[pai]); subir_min(pai)
        senão subir_max(i)
```



Heap min-max

Exemplo 2: Inserção

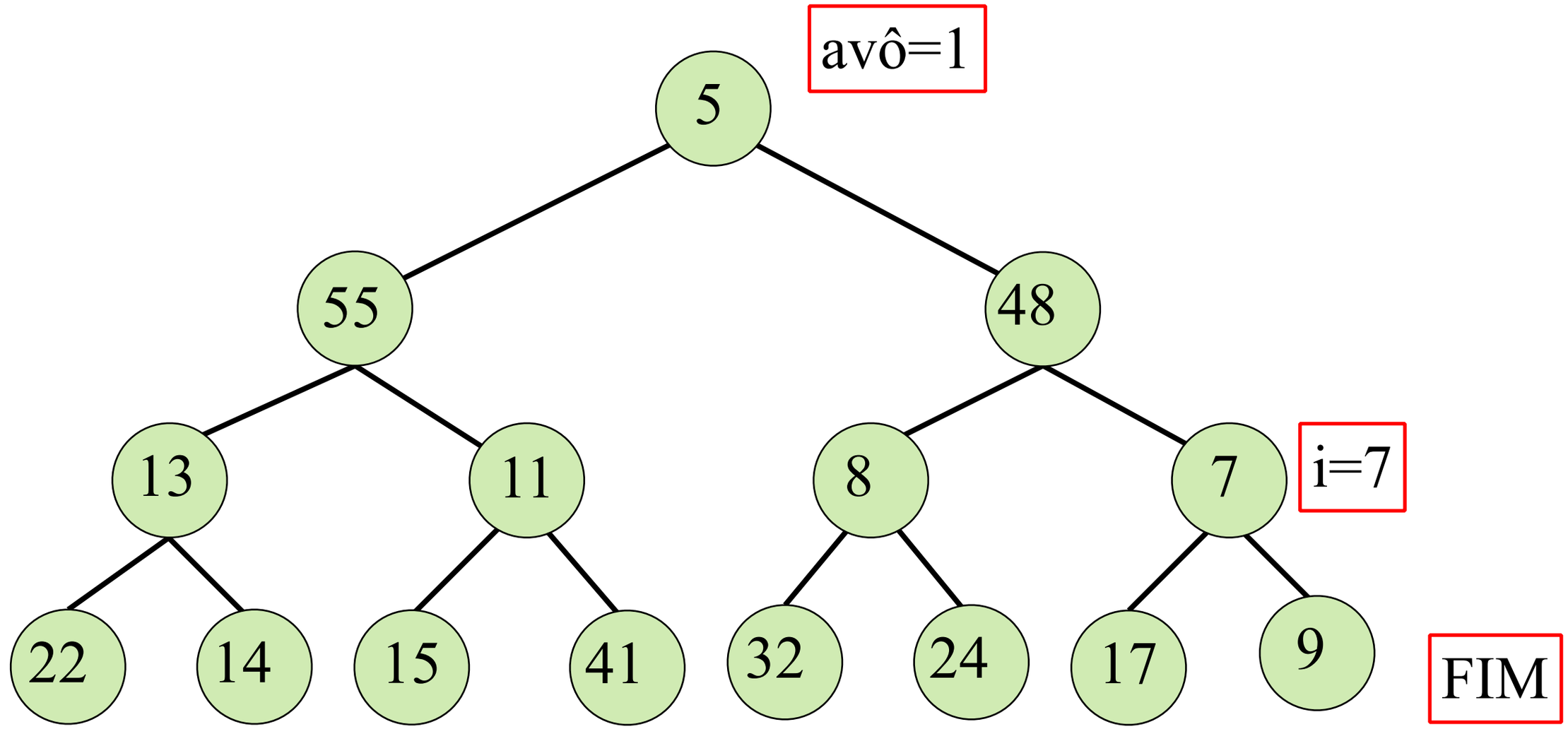
```
subir_min(i)
   se H[i] tem avô então
      se H[i].chave < H[avô].chave então
         Troca(H[i], H[avô])
         subir_min(avô)
```



Heap min-max

Exemplo 2: Inserção

```
subir_min(i)
    se H[i] tem avô então
        se H[i].chave < H[avô].chave então
            Troca(H[i], H[avô])
            subir_min(avô)
```



FIM